Student Number: 202737

1. Introduction

This report explores the application of Convolutional Neural Networks (CNNs) on the CIFAR-10 image data-set. Different hyperparameters / network structures are explored and analysed using accuracy / loss error metrics. The CIFAR-10 data-set contains 32x32 RGB images. There are 10 classes to predict: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck. There are 60,000 images in total, 50,000 allocated as training and 10,000 allocated as testing. All images contain class labels, the distribution of image classes is balanced.

2. Approach

Prior to building models data augmentation was applied. Data augmentation is a typical pre-processing step in computer vision tasks that is used to create diversity from existing training images. Augmentation was used in 2018 to successfully improve upon the then state of the art error rate on CIFAR-10 [3]. Here I have implemented a 5 degree random rotation, 0.5 probability of a horizontal flip, and a random crop where images are kept 32x32 but a padding of 2 is introduced. The training data was further split using a 0.9 ratio into training (45,000) and validation (5,000) sets. Validation set was transformed using the test set to avoid augmentation. This allows our validation loss / accuracy to best represent our unseen test sets. A base model was used as a benchmark for accuracy / loss to compare with subsequent models. The base model I chose contained 3 hidden layers: 2 convolutional layers and one fully connected layer. All models were chosen to use Kaiming Normal weight initialisation [11] for their convolutional layers and Xavier normal [17] for their fully connected layers. The batch size was initially taken as 64. The activation function chosen was Rectified Linear Units (ReLU) [4] and the optimization technique was adaptive moments (adam) [2]. The loss function used was Cross Entropy [20] which calculates number of bits required to represent the average event from one distribution compared to another distribution. The kernel size for our base model was 3x3, Max Pooling on our base model used a kernel size of 2x2. Batch normalisation [15] was also included after our first hidden layer to theoretically improve the stability of our model. Dropout of 0.5 was introduced prior to our fully connected layer to help avoid overfitting [13]. The hyperparameters / network structures I explored following this base model include: batch size, optimization technique, average pooling, wider network, deeper network, dropout rates. Finally the AlexNet [1] architecture was implemented to allow for a comparison between this seminal model and my own models. Each model was run for 20 epochs. Learning rate was chosen using a modified version of the Learning Rate finder [16].

3. Methodology

In total 10 models were deployed, models were trained on training data and evaluated on validation data using simple functions built over PyTorch. First The information contained in our iterator objects was sent to our GPU device to expedite model training. Initially the gradient of our optimizer is set to 0. Class predictions are made using the initialised weights; loss and accuracy is recorded and it is the loss that is used to compute the gradient of the current tensor via the chain rule. Finally the value of these gradients are passed to our optimizer, then using PyTorch step function gradient descent occurs. Evaluating the current loss against the prior loss is what allows models to learn the optimal weights for each piece of data it is presented. It should be noted that any hyperparameters / network structures explored that improved (decreased) validation loss were implemented in subsequent models. During training and evaluation the validation loss at current epoch is evaluated against the best validation loss from previous epochs which is updated if the loss is smaller. The weights from this loss are saved to a system state dictionary to be used at test time.

3.1. Network Structure

CNNs were first introduced by Yann LeCun [18] continuing the works of Kunihiko Fukushima [10]. The first model - LeNet, had 7 layers, and was able to recognize handwritten digits. AlexNet [1] was introduced in 2012 and has galvanized much research since. Image Transformers are models that recently overtook CNNs being the most accurate at predicting CIFAR-10 [7].

Convolutional Layers refer to the layers within a CNN that are used to extract features from our input data. A layer size and kernel size are specified. This kernel size effectively becomes our filter which is slid over our layer size taking the dot product between our filter and the different parts of our input image; the output is known as a feature map. It is at this stage that our CNN starts to extract features used for classification. In the first layer features like lines can be obtained, as we introduce more convolutional layers more complex features can be obtained.

Pooling Layers refers to the layers used to down sample a convolutional layer. They helps prevent against overly precise positioning of feature maps that cannot be generalized to unseen images, thus improving feature extraction. Effectively, images become lower resolution which creates a larger area for our feature representations to be contained within. Max pooling [9] refers to returning the maximum value of our RGB pixel intensities for each section of our feature map contained within our filter. Average pooling [14] refers to returning the average value of our RGB pixel intensities for each section of our feature map within our filter. This research builds models using both Max pooling and Average pooling both with kernel size 2x2.

Fully Connected Layers refer to the layers implemented after convolving and pooling is conducted. They map the inputs from one layer to every activation unit of the next layer. By decreasing our output dimensions in each fully connected layer we are telling our model to reduce the set of features used for classification and return only the most import ones. Fully connected layers consist of weights, biases and neurons.

3.2. Hyperparameters

Batch Size refers to the number of samples processed by our model prior to the loss being updated. Typically, smaller batch sizes require more training epochs as it means our model makes smaller gradient updates compared to larger batch sizes. This research explores 3 values for batch size: 64, 128, 256.

Batch normalization [15] refers to the method of normalizing the mean and variance output of convolutional layers. Normalizing refers to creating a uniform distribution. This process works to stabilise the learning process as outputs are smoothed which simplifies our optimization function that is solved to update weights. This research uses Batch normalization equal to the output of our convolutional layer.

Dropout refers to the method of randomly dropping a proportion of layer output, effectively forcing nodes to take on more or less weight for the inputs. Dropout is implemented prior to building fully connected layers. Theoretically, increasing the dropout ratio should lower the chances of overfitting [13]. The units that are retained using probability p contain outgoing weights that are multiplied by p at test time. All models unless stated otherwise use dropout ratio of 0.5. Our research builds models to explore 3 values of dropout: 0.2, 0.5, 0.8.

Optimizers refers to the mathematical algorithms that are used to update network weights during iterative training. The classical optimization technique is Stochastic Gradient Decent (SGD) [8], which computes step size as a function - the gradient of the loss function multiplied by the learning rate. The new parameters in SGD are equal to the old parameters minus the step size. Adam [2] is an extension of SGD that introduces momentum similar to that used in RMSprop [4]. Estimation of both first and second order moments provides an optimization technique that can handle sparse gradients on stochastically noisy problems. This means it can be applied to many problems where large data-sets or high model parameters are in effect. This research builds a model to explore SGD but primarily uses adam.

Activation Function refers to the function that is used to map the outputs of one layer to the inputs of another layer. The activation function is the means by which non-linearity is introduced into our models. Here we consider Rectified Linear Units (ReLU) [4] activation which maps inputs to outputs as: f(x) = max(0,x), and was introduced to mitigate the problem of vanishing/exploding gradients [6]. This research focuses on ReLU activation, although alternatives like Leaky ReLU, Tanh or Sigmoid could be considered.

Weight Initialisation in recent years has become more applied to the choice of activation function used. Effective weight initialisation protects layer activation outputs from exploding or vanishing. In our case we are using ReLU and our weight initialisation techniques are Kaiming normal [11] and Glorot (Xavier) normal [17]. This research does not explore different weight initialisation techniques.

Kernel Size refers to the pixel width x height for our filter. Filters act as a feature extractor, they slide over our feature map given a specified stride and padding extracting unique features contained within these sub-areas of our training images. Unique features help to classify on unseen test sets. This research only uses kernel size 2x2.

Width refers to the number of channels within each convolutional layer. Increasing the width in the layers also increases the number of parameters (weights, biases) contained within the model, which should help us to extract more complex features. Two different width architectures are explored: Initial input width with 32, and 64.

Depth refers to the number of hidden layers within the model. Increasing the number of hidden layers within the model increases the number of parameters contained within the model; useful for extracting more complex features. Two depths are explored in this research: 3 hidden layers, 5 hidden layers.

Learning Rate is the most important hyperparameter and refers to the value that is used by the optimizer to determine the step size at each iteration. Small learning rates can get caught in local minimums, large learning rates may not be able to converge to a global minimum due to a large step size. Determining the optimal learning rate for this research is implemented by using a range finder function [16] within our learning rate finder class. We pass in the model, optimizer and criterion to the class. The range finder then instantiates a learning rate object which iterates over batches of the training data within specified learning rate bounds. The smallest learning rate passed to the range finder is exponentially increased until our loss diverges. The point at which the loss curve begins to flatten indicates that the learning rate is beginning to grow too large to provide optimal global loss convergence. Optimal learning rate is taken to be one order smaller than the point at which the loss curve flattens. See Appendix A for table of each model's learning rate.

4. Results and Discussion

The results table displayed below lists the overall accuracy of each model based on predictions on our CIFAR-10 image data-set. G-gap stands for generalization gap. See Appendix B, C. for graphs of validation loss / accuracy.

Model Results				
Model	Train	Val	Test	G-gap
BaseModel	62.39	70.02	70.07	-7.68
Batch128+	68.48	73.71	73.71	-5.23
Batch256	67.86	73.45	73.51	-5.65
SGD	41.64	46.86	47.43	-5.79
AVGPool+	71.56	76.48	76.54	-4.98
Wider+	79.75	81.13	80.68	-0.93
Deeper+	84.58	84.75	85.22	-0.64
Dropout(0.2)	90.96	84.94	84.04	6.92
Dropout(0.8)	77.60	79.79	78.57	-0.97
AlexNet	74.80	75.53	74.97	-0.17

Figure 1. All scores are denoted in percentage. Train accuracy is the corresponding score to the best validation accuracy. Generalization gap refers to the difference between train and test accuracy. A + next to a model denotes an improved test score and implementation of this parameter in subsequent trials

Out of the three batch sizes tested on our base model 128 performed the best on validation and test sets. Increasing the number of epochs may have allowed the base model (batch size 64) to have continued learning although this was not tested. The worst performing model was SGD; failing to account for momentum when updating the gradient of the loss function. Average pooling increased model accuracy by 2.83% compared with max pooling. Max pooling returns the brightest pixel over a specified area, useful for black and white images where sharp changes in pixel intensities represent features, like those found in the MNIST data set. Less effective here on our image set containing RGB pixels.

Implementing a wider network increased the number of parameters in the models (See appendix A). As mentioned previously, increasing the model parameters should allow models to learn more complex features from the data. This theory is backed up by the improvement in accuracy as shown in figure 1. It should also be noted that this improvement in accuracy was also met with a deterioration in generalization gap. Although the g-gap is still negative (i.e., testing accuracy still higher than training), it could be a sign that our model is starting to overfit [19], even with dropout implemented.

Our deeper network implemented an extra convolutional and fully connected layer. This implementation significantly increased the number of parameters contained within the model (See appendix A). Again, looking at the table the increase in parameters translated to an increase in accuracy on all three data sets. The generalization gap was still negative but worsened again from our wider network.

To showcase the efficacy of dropout as a means to prevent overfitting our deeper network was altered from using 0.5 to 0.2. Less neurons being dropped means more information is passed through the fully connected layers. Inspecting train and val accuracies may lead us to purportedly assume that this model is better compared with our deeper model. However, its test accuracy was lower and its generalization gap was 6.92 which was the only instance where train accuracy was lower than test; a sign of overfitting. The graph in Appendix D illustrates the overfitting of this models train/val accuracy over 20 epochs.

Our dropout of 0.8 underperformed compared with our wider model, which had significantly less parameters. Too many neurons being dropped doesn't allow enough useful information to be passed on to be used for classification.

AlexNet is model that uses max pooling and dropout. It contains 23,272,266 parameters and achieved an accuracy of 74.97% on our test set. Our basemodel using average pooling achieved better accuracy with significantly less parameters. My models used batch normalization where AlexNet does not. Hopefully these results and discussion elucidate the strength of advanced regularisation strategies for increasing accuracy and avoiding overfitting. Increasing model parameters does not necessarily improve model performance and therefore all regularisation techniques should be considered when building a model for any image datasets.

Due to the complexity of CNN architectures, an almost endless list of hyperparameters / network structures could've been explored and cross-examined to try and improve accuracy. Average pooling improving accuracy on batchsize 128 does not necessarily mean that it would perform optimally for other batch sizes. As such, each network update means all previously explored hyperparameters could be re-explored e.g. re-testing batchsize for average pooling.

Some areas of further research I am interested in are applying optimizer RMSprop and utilising hyperparameter tuning techniques for each of my models. RMSprop was introduced by Hinton [4] and first implemented by Graves [5]. As it uses a decaying average of partial gradients similar to adam, with fine tuning I believe it could be a competitive alternative to adam. Other than our LR finder, hyperparameters were not tuned for each model. Although individual use cases provided a clear analysis into the effects of each hyperparameter, accuracy may have improved by using techniques like grid search or bayesian optimization [12].

References

[1] G. E. H. Alex. Krizhevsky, Ilya. Sutskever. Imagenet classification with deep convolutional neural networks, 2012. 1

[2] J. B. Diederik P. Kingma. Adam: A method for stochastic optimization, 2014. 1, 2

[3] D. M. V. V. Q. V. L. Ekin. D. Cubuk, Barent. Zoph. Autoaugment: Learning augmentation policies from data, 2018. 1

[4] V. N. Geoffrey. Hinton. Rectified linear units improve restricted boltzmann machines, 2010. 1, 2, 3

[5] A. Graves. Generating sequences with recurrent neural networks, 2014. 3

[6] S. Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. 2

[7] A. S. G. S. H. J. Hugo Touvron, Matthieu Cord. Going deeper with image transformers. 1

[8] S. ichi. Amari. Backpropagation and stochastic gradient descent method, 1993. 2

[9] G. A. D. C. Jawad. Nagi, Frederick. Ducatelle. Max-pooling convolutional neural networks for vision-based hand gesture recognition, 2015. 2

[10] S. M. K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition, 1982. 1

[11] S. R. J. S. Kaiming. He, Xiangyu. Zhang. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2014. 1, 2

[12] J. Mockus. On bayesian methods for seeking the extremum, 1997. 3

[13] A. K. I. S. R. S. Nitish. Srivastava, Geoffrey. Hinton. Dropout: A simple way to prevent neural networks from overfitting, 2014. 1, 2

[14] B. Sabri. A comparison between average and max-pooling in convolutional neural networks for scoliosis classification, 2020. 2

[15] C. S. Sergey. Ioffe. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015. 1, 2

[16] L. N. Smith. Cyclical learning rates for training neural networks, 2015. 1, 2

[17] Y. B. Xavier. Glorot. Understanding the difficulty of training deep feedforward neural networks, 2014. 1, 2

[18] Y. B. Yann. LeCun. Convolutional networks for images, speech, and time-series, 1995. 1

[19] X. Ying. An overview of overfitting and its solutions. 3

[20] M. S. Z. Zhang. Generalized cross entropy loss for training deep neural networks with noisy labels, 2018. 1

Appendix

Appendix A.

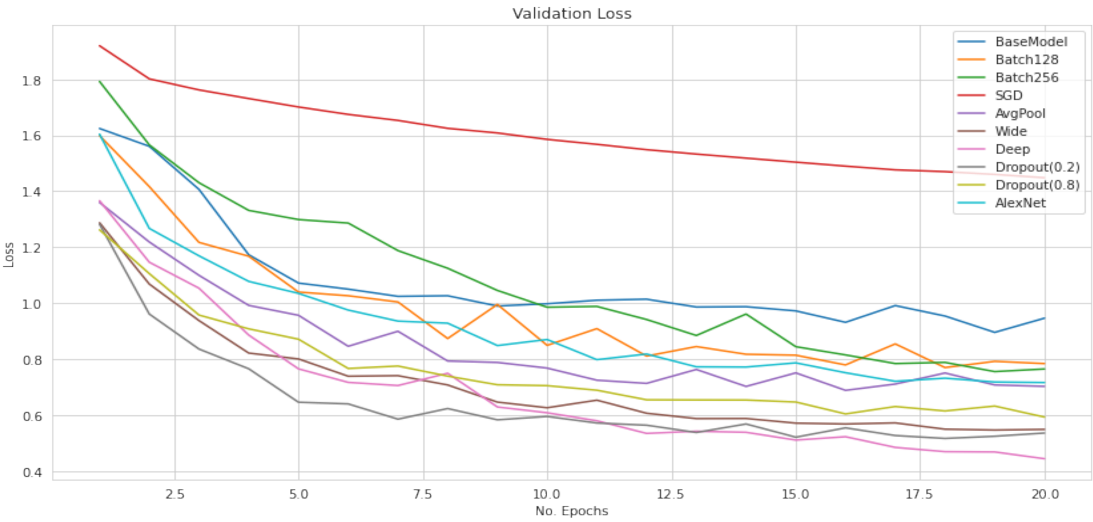
Table: Model – Parameters and Learning Rates

Model	Parameters	Learning Rate
BaseModel	2,159,242	0.01
Batch128	2,159,242	0.01
Batch256	2,159,242	0.01
SGD	2,159,242	0.001
AVGPool	2,159,242	0.005
Wider	8,623,370	0.001
Deeper	36,626,570	0.001
Dropout(0.2)	36,626,570	0.001
Dropout(0.8)	36,626,570	0.001
AlexNet	23,272,266	0.001

Note: Parameters refers to the number of neurons (weights + biases) for each model architecture

Appendix B.

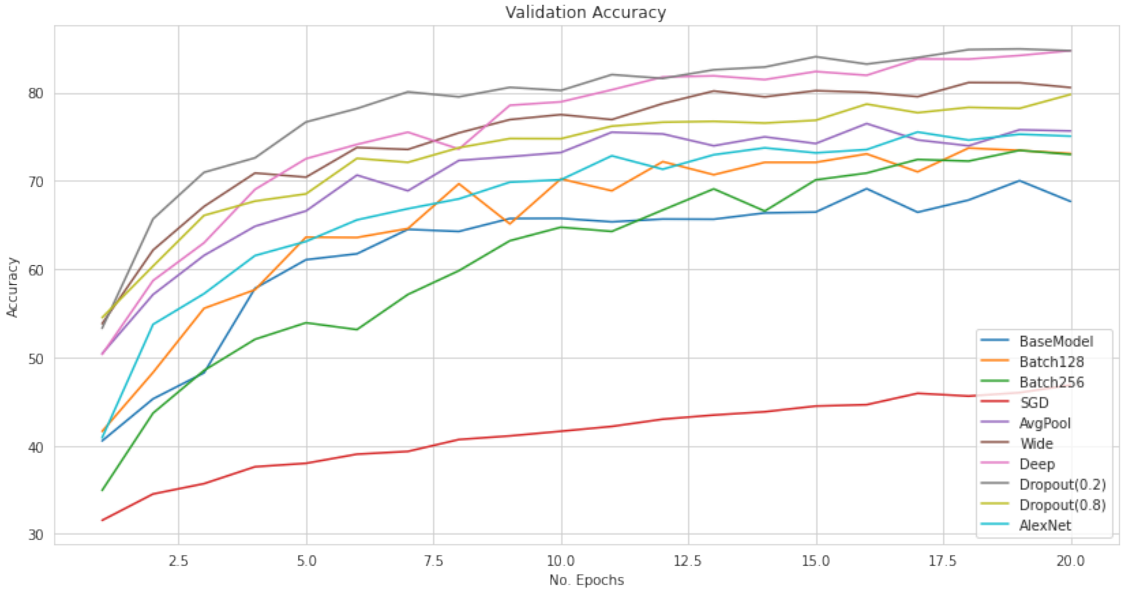
Graph: Validation loss all models



Note: validation loss is based on our loss metric – Cross Entropy

Appendix C.

Graph: Validation accuracy all models



Note: Accuracy is scored as % of correctly guessed image labels

Appendix D.

Graph: Train, Val loss for model Wider + Deep model (0.2) Dropout

Note: Loss score is based on our loss metric – Cross Entropy